Dynamic Traceroute Visualization at Multiple Abstraction Levels*

Massimo Candela, Marco Di Bartolomeo,
Giuseppe Di Battista, and Claudio Squarcella

Dipartimento di Ingegneria, Università Roma Tre, Italy
{candela,dibartolomeo,gdb,squarcel}@dia.uniroma3.it

Abstract. We present a system, called TPLAY, for the visualization of the traceroutes performed by the Internet probes deployed by active measurement projects. These traceroutes are continuously executed towards selected Internet targets. TPLAY allows to look at traceroutes at different abstraction levels and to animate the evolution of traceroutes during a selected time interval. The system has been extensively tested on traceroutes performed by RIPE Atlas [22] Internet probes.

1 Introduction

The *traceroute* command is one of the most popular computer network diagnostic tools. It can be used on computers connected to the Internet to compute the path (route) towards a given IP address, also called *traceroute path*. It is probably the simplest tool to gain some knowledge on the Internet topology. Because of its simplicity and effectiveness, it attracted the interest of several researchers that developed services for visualizing the Internet paths discovered by executing one or more traceroute commands.

Broadly speaking, there are two groups of traceroute visualization systems: tools developed for local technical debugging purposes and tools that aim at reconstructing and displaying large portions of the Internet topology. Several tools of the first group visualize a single traceroute on a map, showing the geo-location of the traversed routers. A few examples follow. Xtraceroute [10] is a graphical version of the traceroute program. It displays individual routes on an interactive rotating globe as a series of yellow lines between sites, shown as small spheres of different colors. GTrace [20] and VisualRoute [30] are traceroute and network diagnostic tools that provide a 2D geographical visualization of paths. The latter also features more abstract representations taking into account other information, e.g. the round-trip time between intermediate hops. In the second group there are several tools (see e.g. [18,5]) that merge the paths generated by multiple traceroutes into directed graphs and show them in some type of drawing.

In recent years the visualization of Internet measurements has seen a growing interest. This is mainly due to the existence of several projects that deploy *probes* in the Internet. Probes are systems that perform traceroutes and other measurements (e.g. ping,

* Partially supported by the ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations" and by the European Community's Seventh Framework Programme (FP7/2007-2013) grant no. 317647 (Leone). We thank RIPE NCC for collaborating to the development of the graph animation framework used in this work.

S. Wismath and A. Wolff (Eds.): GD 2013, LNCS 8242, pp. 496–507, 2013.

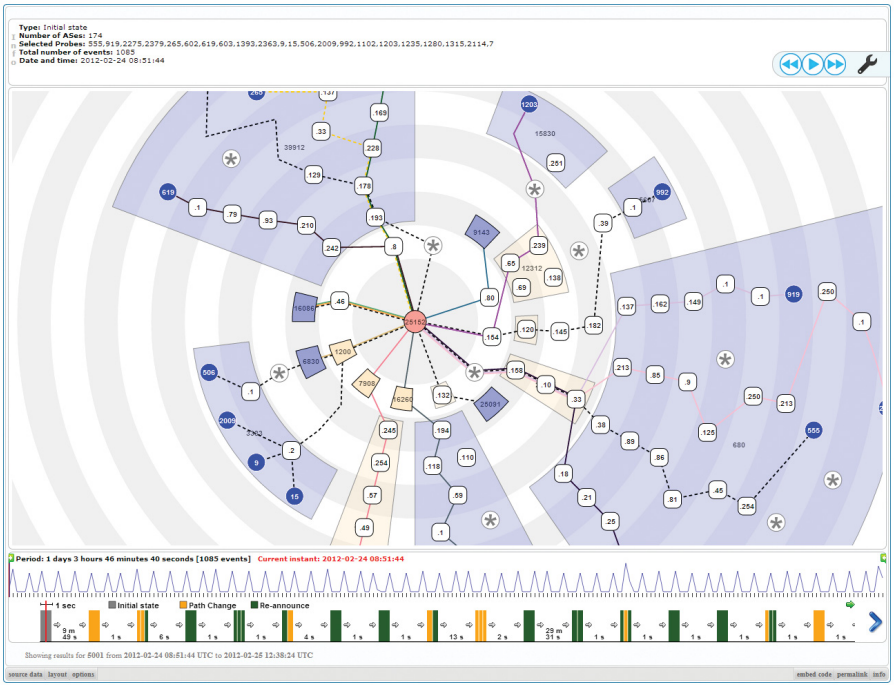


Fig. 1. The main interface of TPLAY

HTTP queries) towards selected targets. They produce a huge amount of data that is difficult to explore, especially when dealing with the network topology. Some examples follow. SamKnows [7] is a broadband measurement service for consumers. MisuraInternet [4] is an Italian project that measures the quality of broadband access. BISmark [28] is a platform for measuring the performance of ISPs. RIPE Atlas [22], CAIDA Ark [2], and M-Lab [3] continuously perform large scale measurements towards several targets.

In this paper we present a system for traceroute visualization called TPLAY, designed for supporting Internet Service Providers (ISPs) and Internet Authorities in the management and maintenance of the network. The requirements were gathered interacting with several ISPs, within the Leone FP7 EC Project, and with the RIPE Network Coordination Center (RIPE NCC). The system works as follows. The user selects a set $\mathcal{S}$ of probes of a certain Internet measurement project (all the experiments in this paper have been conducted using RIPE Atlas [22] probes), a target IP address τ, and a time interval $\mathcal{T}$, and obtains a visualization of how the traceroutes issued by the probes in $\mathcal{S}$ reach τ during $\mathcal{T}$. TPLAY can be used to study several properties of traceroute paths. These include assessing the reachability of τ over time, discovering the ISPs that provide connectivity to reach it, monitoring the length of traceroute paths as a performance indicator, and inferring how routing policies affect the paths of different probes in $\mathcal{S}$.

A snapshot of TPLAY is in Fig. 1. The routing graph is presented with a radial drawing. The geometric distance between τ and any object reflects the topological distance of that object in the network. Also, since traceroutes tend to give too many details, the

system allows to look at the network at different abstraction levels. Finally, the evolution of traceroute paths over time is presented by means of geometric animation.

The paper is organized as follows. In Section 2 we detail the use cases, describe the adopted visualization metaphor, and introduce some formal terminology. In Section 3 we detail the algorithms used to compute the visualization and compare them to the state-of-the-art. In Section 4 we describe the prototype implementation of our tool and the technical challenges we faced. Section 5 contains conclusions and future directions.

2 Use Cases and Visualization Metaphor

The main tasks associated with our system are detailed below. Two of them deal with *Autonomous Systems* (ASes), i.e. entities representing Internet administrative authorities. Once the input is specified as detailed in Section 1, the user is interested in the following. *Security*: knowing what ASes provide connectivity to reach the target over time. That is interesting from the perspective of security, because some ASes may be less trusted than others. *Policy*: seeing how traffic is routed inside a specific AS over time. That helps discover load balancing issues or differences in the routing applied to different probes. *Distance*: knowing the number of hops traversed by each probe over time. Longer paths are indeed potentially responsible for instability and inefficiency. *Dynamics*: seeing how the routing changes at a specific time instant, based on external key indicators. For example, the user may want to check if the routing has changed after a noticeable drop in the round-trip delay experienced when reaching the target.

We discarded solutions based on geographic representations for many reasons. First of all, the fact that a router belongs to a certain ISP or AS is the main piece of information for our purposes, whereas geography is only a secondary feature that further characterizes the nodes in the network. Also, the geo-location data associated with IP addresses is often wrong or missing, and anycast addresses (i.e., those assigned to more than one physical device) can not be mapped to a single location. Finally, the use of landmarks on geographical maps would require special care to avoid geometric cluttering. Motivated by the above, we focused on a topological representation of the data.

The visualization metaphor we adopted is presented below together with supporting motivations. Graphs are represented with *radial layered drawings*, where vertices are placed on concentric circles and targets are in the center. This style of drawing is notably effective for visualizing sparse hierarchical graphs (see, e.g., [31]); in Section 3 we show that our application domain meets such requirement. The probes originating the traceroutes are in the periphery of the drawing. This approach is effective in displaying topological distances. Moreover, radial drawings have their center as the only focus point, which avoids giving probes additional importance due to a privileged geometric position. Finally, the drawing looks like an abstract geography and hence borrows the typical user experience deriving from cartography and geographical visualization.

The need of visualizing the network at different abstraction levels is met by partitioning the set of routers into *clusters*. In our setting, clusters are in correspondence with ASes. The user can modify the representation by interacting with any cluster to either contract or expand it. A *contraction* causes all the routers in the cluster to be merged into a single object representing the cluster, while an *expansion* does the opposite. Collapsing all clusters leads to a high-level, uncluttered view of the graph. On the other

hand, the user can expand all the clusters to see all the traversed routers. In general, the user can arbitrarily expand any subset of clusters to examine them in detail.

Paths for reaching the target from the probes change over time. A natural way to show the evolution of traceroutes at different time instants is to present an animation of the drawing. More precisely, for each instant in a given time interval we show a different drawing, corresponding to the traceroutes that are available at that instant. We animate the change from a drawing to a successive one by means of a geometric morph.

Since the visualization is highly interactive and the graph changes over time, preserving the mental map is of paramount importance. Indeed, the user can both animate the drawing in a specific time interval and expand/contract individual clusters. We require that the same drawing is visualized for any two sequences of cluster expansions/contractions that produce the same graph. Also, the graph should be animated smoothly, even at the expense of traversing drawings that are not aesthetically optimal.

Traceroute paths cannot simply be merged and displayed in an aggregate fashion, since each of them has its own informative value and can change over time. For this reason, we represent paths adopting a metro-line metaphor [24] and draw them using different colors. Further, paths that never change in the selected time interval should be easily distinguished. In this context we adopt the method described in [14]. Paths that do not change are partitioned into sets such that each of them determines a tree on the graph. Each tree is depicted with dashed lines and a distinctive color. This has the effect of reducing the number of lines in the drawing, while preserving the routing information for each probe. Paths that change are instead represented by solid lines.

The objects to be visualized are formally defined as follows. Consider a time interval $\mathcal{T}$ and a set of probes $\mathcal{S}$. During $\mathcal{T}$ each probe periodically issues a traceroute towards a target IP address τ. A *traceroute* from probe $\sigma \in \mathcal{S}$ produces a simple directed path on the Internet from σ to τ. If such a path is available in Internet at time $t \in \mathcal{T}$, then it is *valid* at time t. Each vertex of a traceroute originated from $\sigma \in \mathcal{S}$ is either a router or a computer. Vertices are identified as follows: (1) σ has a unique identifier selected by the RIPE NCC; (2) vertices with a *public* IP address are identified by it; (3) vertices with a *private* IP address are identified by a pair composed of their address and the identifier of σ; (4) the remaining vertices are labeled with a "*" (i.e. an unknown IP address). For the sake of simplicity, consecutive vertices labeled with "*" are merged into one. A vertex labeled with "*" is identified by the identifiers of its neighbors in the traceroute.

A digraph G_t is defined at each instant $t \in \mathcal{T}$ as the union of all the paths valid at t produced by the traceroutes issued by the probes of $\mathcal{S}$. A digraph $G_{\mathcal{T}}$ is defined as the union of all graphs G_t. Each vertex of $G_{\mathcal{T}}$ is assigned to a *cluster* as follows. (1) Each probe is assigned to the cluster that corresponds to the AS where it is hosted. (2) Each vertex identified by a public IP address [6] is assigned to a cluster that corresponds to the AS announcing that address on the Internet. This information is extracted from the RIPEstat [23] database and may occasionally be missing. (3) Each vertex v that is not assigned to a cluster after the previous steps is managed as follows. Consider all traceroute paths containing v. For traceroute p let μ (ν) be the cluster assigned to the nearest predecessor (successor) of v with an assigned cluster. If $\mu = \nu$ then μ is added to the set

of candidate clusters for v. If such set has exactly one cluster, v is assigned to it. If there is more than one candidate, an inconsistency is detected and the procedure terminates prematurely. (4) Each remaining vertex is assigned to a corresponding *fictitious* cluster. We define V_μ as the set of vertices assigned to cluster μ.

For any $t \in \mathcal{T}$ G_t can be visualized at different abstraction levels. Namely, the user can select a set $\mathcal{E}$ of clusters that are fully visualized and each cluster that is in the complement $\bar{\mathcal{E}}$ of $\mathcal{E}$ is contracted into one vertex. More formally, given the pair $G_t, \mathcal{E}$ the visualized graph $G_{t,\mathcal{E}}(V, E)$ is defined as follows. V is the union of the V_μ for all clusters $\mu \in \mathcal{E}$, plus one vertex for each cluster in $\bar{\mathcal{E}}$. E contains the following edges. Consider edge (u, v) of G_t and clusters μ and ν, with $u \in \mu$ and $v \in \nu$. If $\mu \neq \nu$, $\mu \in \mathcal{E}$, and $\nu \in \mathcal{E}$, then add edge (u, v). If both μ and ν are in $\bar{\mathcal{E}}$ then add edge (μ, ν). If $\mu \in \mathcal{E}$ ($\mu \in \bar{\mathcal{E}}$) and $\nu \in \bar{\mathcal{E}}$ ($\nu \in \mathcal{E}$) then add edge (u, ν) ((μ, v)). We define $G_{\mu,t}$ as the subgraph of G_t induced by V_μ. Analogously, we define $G_{\mu,\mathcal{T}}$ as the subgraph of $G_\mathcal{T}$ induced by V_μ. We define $G_{\mathcal{T},\mathcal{E}}$ as the union of the $G_{t,\mathcal{E}}$ for each $t \in \mathcal{T}$.

Fig. 1 shows an overview of our prototype implementation. Let $t \in \mathcal{T}$ be the time instant selected by the user. Graph $G_{t,\mathcal{E}}$ is represented by a radial drawing centered in τ. All vertices and clusters that appear in at least one traceroute in $\mathcal{T}$ are in the drawing, including those that are not traversed by any traceroute at time t. Probes in $\mathcal{S}$ are represented as blue circles and labeled with their identifier. Vertices are represented as white rounded rectangles and labeled with the last byte of their IP address, or with a "*". Clusters are represented as annular sectors and labeled with their AS number. Note that vertices assigned to expanded clusters are enclosed in their sectors, while sectors of contracted clusters are empty. The light red cluster contains τ. Clusters containing probes in $\mathcal{S}$ are light blue. The remaining clusters are light yellow. Fictitious clusters are not displayed. Each path from a probe $\sigma \in \mathcal{S}$ to τ is represented by a colored curve from σ to τ passing through all intermediate vertices. Paths are either solid or dashed, depending on whether they change or not during the time interval $\mathcal{T}$. Concentric circles in the background represent the increasing topological distance of vertices.

Fig. 2 contains various details on how the interaction with the visualization works. A graph with static paths and no expanded clusters is presented in Fig. 2(a). It is related to a target τ, a set of probes $\mathcal{S}$, and a small time interval $\mathcal{T}'$. Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. A graph for τ, $\mathcal{S}$ and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$) is presented in Fig. 2(b). Some dynamic paths are visible. The same graph is presented in Fig. 2(c) with one expanded cluster. Note how the ordering of clusters and vertices on the radial layers is preserved. Fig. 2(d) shows the same expanded graph at a different time instant. The intermediate vertices of two paths are different.

Fig. 2 also helps us explain how the tasks detailed at the beginning of the section can be accomplished. The *Security* task is satisfied in Fig. 2(a): we can see how ASes 1200 and 20965 provide connectivity to reach the target. The *Policy* and *Distance* tasks are addressed in Fig. 2(c), where the length and structure of the paths from each of the three probes 619, 602, 265 is clearly visible. The *Dynamics* task is solved in Figg. 2(c)-(d), where we can see how the paths change for probes 619 and 602 after a routing event.

The user interaction plays a major role in our metaphor. The reader can visit [8] for an example video of the interaction with TPLAY.

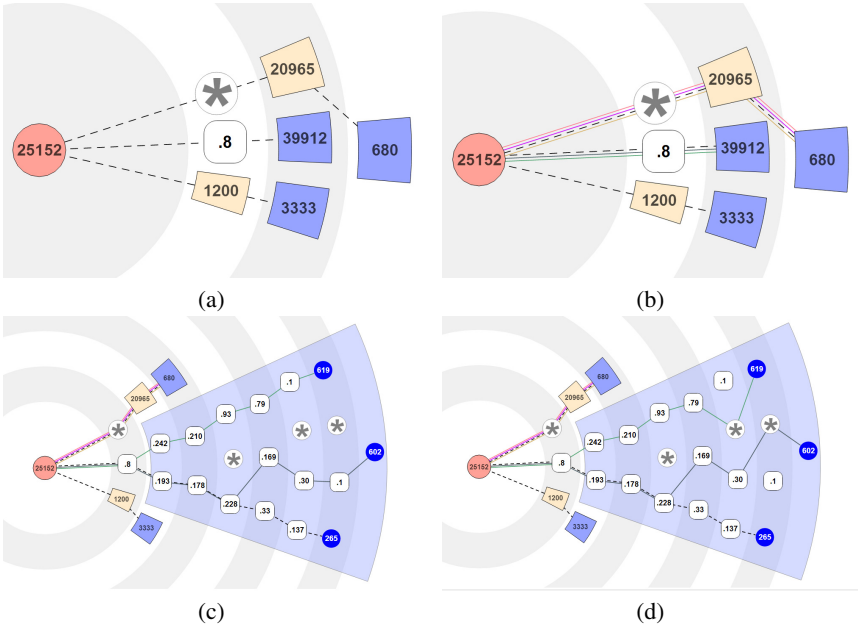
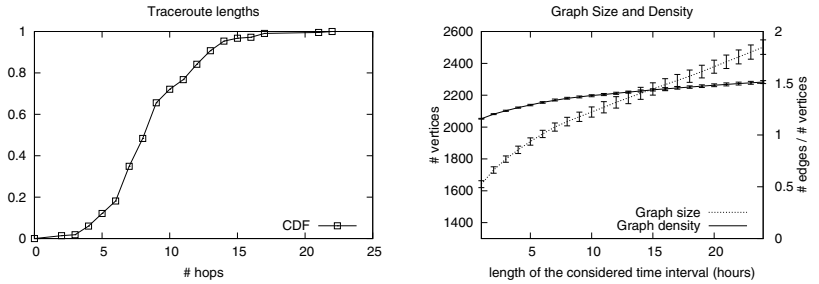


Fig. 2. Details of the interactive features of our visualization. (a) A graph $G_{\mathcal{T}'}$ relative to a target τ, a set of probes $\mathcal{S}$, and a time interval $\mathcal{T}'$. All paths in $G_{\mathcal{T}'}$ are static and all clusters contracted. (b) A graph $G_{\mathcal{T}''}$ relative to τ, $\mathcal{S}$, and $\mathcal{T}''$ ($|\mathcal{T}''| > |\mathcal{T}'|$). Some paths are dynamic and all clusters are contracted. (c) $G_{\mathcal{T}''}$ with an expanded cluster. (d) $G_{\mathcal{T}''}$ at a different time instant.

3 The Algorithms

We started our analysis by computing several statistics on the RIPE Atlas data set that we used to test the system. It consists of traceroutes executed in one month (July 2012) by 200 probes. Fig. 3 presents the main results of our analysis. In Fig. 3(a) we plot a cumulative distribution function of the length of traceroute paths. That gives us a rough indication on the maximum distance between a probe in $\mathcal{S}$ and τ. The plot shows that traceroutes with more than 15 vertices are rare, confirming the suitability of the radial metaphor. In Fig. 3(b) we plot the number of vertices and the density ($|E|/|V|$) of $G_{\mathcal{T}}$ as a function of $\mathcal{T}$. It turns out that $G_{\mathcal{T}}$ is quite sparse for time intervals that are compatible with the application domain. In particular, the density ranges between 1.2 and 1.5 for time intervals within 24 hours. The number of vertices is in the range of 2000.

As a second step, we performed experiments using spring embedders and hierarchical drawing algorithms. Layouts produced by spring embedders [29] are unsuitable for our metaphor, because the topological distance between vertices is not always represented and because they produce drawings with not enough regularity. Also, they tend to introduce crossings that are avoidable, for the expected density of the data set. For hierarchical drawing, we experimented both basic algorithms [29] and variations that allow to represent clustered graphs [25,26]. The experiments put in evidence that crossing-reduction heuristics like those in [25,26] are quite effective. However, in our



(a) (b)

Fig. 3. Statistics on the data set. (a) Cumulative distribution function (CDF) of the length of traceroute paths at July, 1st 2012 at 00:00. CDFs at different instants exhibit similar features. (b) Plot showing the number of vertices and the density of $G_{\mathcal{T}}$ as a function of $\mathcal{T}$. For each day in the month we set an initial time at 00:00 and grow $\mathcal{T}$ from 1 to 24 hours. For each value of $\mathcal{T}$ we plot the average density and number of vertices. We report the standard deviation with error bars.

case most graphs are planar or quasi-planar and hence planarity-based methods are more attractive. Finally, we discarded upward planar drawings [29]. The main reason is that they tend to use vertical space to resolve crossings, which may result in large geometric distances between vertices that are topologically close.

A very high level and informal description of our algorithmic framework is the following. We pre-compute a hierarchical drawing Γ_0 of $G_{\mathcal{T}}$ that integrates all the traceroutes in $\mathcal{T}$. In that drawing all clusters are expanded. The layout is computed in such a way to have few crossings involving connections between clusters. The quality of the layout inside the clusters is considered with lower priority. Moreover, the quality of the drawing of edges that are part of many traceroutes in $\mathcal{T}$ is privileged among the edges of $G_{\mathcal{T}}$. The drawing computed for each cluster is stored and reused in any drawing where that cluster is expanded. The hierarchical drawing is mapped to a radial drawing with a suitable coordinate transformation. Changes in the drawing due to an expansion or contraction of a cluster or a change in traceroutes are visualized with an animation. At any instant $t \in \mathcal{T}$ only the traceroutes that are valid in t are displayed.

For our purposes an interesting reference is [11] that constructs radial drawings adapting techniques of the Sugiyama Framework, but, unfortunately, it does not deal with clusters. The algorithm in [16], which extends the one described in [15], inspired part of our work. However, it proposes a clustered planarity testing algorithm, while we rather need an algorithm for clustered graph planarization, and [16] is not easily extensible for this purpose (neither is the algorithm in [12] that is not suitable for hierarchical drawings). For these reasons we devised a new algorithm to produce clustered hierarchical drawings, as a planarization-oriented variation of [16]. In [21] an algorithm is proposed for the expansion/contraction of clusters of hierarchical drawings, building on [27]. Unfortunately it uses local layering for vertices, while global layering [25,26] is more suitable for our needs because it produces more compact drawings. Indeed a very common use case of TPLAY is to expand all clusters along one or more traceroutes. Local layering would visualize far from τ also vertices in unrelated paths because of the increased need for vertical space of their layers. For this reason we devised a new algorithm for

expanding/contracting clusters that is based on global layering. Differently from [21] it is not a local update scheme, i.e. it computes a new drawing for the whole graph at each interaction. The lower time efficiency is negligible because the graphs commonly handled by TPLAY are small. Finally, mental map preservation during expansion/contraction of clusters is addressed by a geometric morph, implemented as an animation of objects from their initial position to their final position (see, e.g., [14]).

What follows gives more details on our the algorithmic framework. In a preprocessing step several information are computed on $G_{\mathcal{T}}$ that will be used for actual drawings. Given any $G_{\mu,\mathcal{T}}$, a vertex is a *source* (*sink*) of $G_{\mu,\mathcal{T}}$ if it is the last (first) vertex of $G_{\mu,\mathcal{T}}$ encountered in some traceroute path. Each graph $G_{\mu,\mathcal{T}}$ is augmented with extra vertices and edges so that all the longest paths from a source to a sink have the same length. The added vertices are called *fictitious vertices* of μ and ensure that, given an edge $(u, v) \in G_{\mathcal{T}}$, $u \in \mu$, $v \in \nu$, $\mu \neq \nu$, clusters μ and ν do not share a layer in any drawing of $G_{t,\mathcal{E}}$. Moreover, they force edges that leave a cluster by spanning several layers to be routed inside that cluster. A *μ-drawing* is pre-computed for each $G_{\mu,\mathcal{T}}$. It consists of 1. assigning vertices to layers so that all edges are between consecutive layers and 2. computing a total order for the vertices of each layer. A partial order $\prec$ is computed for clusters, such that for any two clusters μ and ν with $\mu \prec \nu$, the vertices of μ appear to the left of the vertices of ν for any drawing Γ where μ and ν share one or more layers. This helps preserve the mental map during expansions/contractions. The preprocessing step requires to compute a drawing Γ_0 of $G_{\mathcal{T}}$ with all clusters expanded. Γ_0 gives the information needed to compute a μ-drawing for each cluster and a partial order $\prec$ for clusters. The algorithm to compute Γ_0 is similar to that in [16], where a PQ-tree [13] is used to order vertices along the layers of the drawing. Our PQ-tree is initialized with a spanning tree of $G_{\mathcal{T}}$ and incrementally updated with the remaining edges that induce ordering constraints. An edge is added only if it does not produce a crossing (i.e. the PQ-tree does not return the null tree). A rejected edge will produce crossings in Γ_0. Edges are added with priority given by their aesthetic importance: namely, they are weighted by the number of traceroutes that traverse them in $\mathcal{T}$. As an implementation detail, we actually compute a total order for clusters to represent a partial order $\prec$. Such an order is produced by a DFS visit of the spanning tree of $G_{\mathcal{T}}$. The tree has an embedding induced by the layer orders produced by the PQ-tree algorithm, and children of a vertex are visited in clockwise order. Intuitively, we preserve the geometric left-to-right order for clusters from Γ_0, and reuse it to produce a drawing of any $G_{t,\mathcal{E}}$.

The computation of the drawing $\Gamma_{\mathcal{T},\mathcal{E}}$ of $G_{\mathcal{T},\mathcal{E}}$ is detailed below. Before that, note that once $\Gamma_{\mathcal{T},\mathcal{E}}$ is computed, we display, for any $t \in \mathcal{T}$ all the vertices of $G_{\mathcal{T},\mathcal{E}}$ but only the edges of $G_{t,\mathcal{E}}$. This is done to preserve the mental map of the user, using $\Gamma_{\mathcal{T},\mathcal{E}}$ as a "framework" that "hosts" the drawings of each instant. First, a layering of $G_{\mathcal{T},\mathcal{E}}$ is computed such that for each vertex the distance from τ is minimized. Also, dummy vertices, called *fictitious vertices* of $G_{\mathcal{T},\mathcal{E}}$, are added so that each edge spans two consecutive layers. Vertices are horizontally ordered on each layer such that: 1. $\prec$ is enforced; 2. for each cluster μ of $\mathcal{E}$, the orders on the layers of its μ-drawing are enforced; 3. the fictitious vertices of $G_{\mathcal{T},\mathcal{E}}$ are placed in such a way to have few crossings. In particular, they must not be interleaved with the vertices of any cluster, that is, the vertices of each cluster must be consecutive on every layer. For this reason, each

fictitious vertex is assigned to a new fictitious cluster, which is inserted in the partial order $\prec$ in an intermediate position between the endpoints of the edge it belongs to. Finally, the ordered layers are used to assign geometric coordinates to vertices. The width of each cluster μ is computed as follows. Consider the layer containing the largest number of vertices assigned to μ. The cluster is assigned a width proportional to this number. Vertices of μ are assigned horizontal coordinates such that they can be enclosed by a rectangle with height proportional to the number of layers assigned to the vertices of μ and width equal to the width of μ. We avoid intersection between enclosing rectangles by means of an auxiliary directed acyclic graph where vertices are clusters of $G_{\mathcal{T},\mathcal{E}}$ and edges are selected from $\prec$ depending on which pairs of clusters share a layer in the current layering of $G_{\mathcal{T},\mathcal{E}}$. Edges are weighted based on the widths of the clusters they are incident to. The total width of the drawing is given by the longest path in this graph. The above is applied recursively to compute the horizontal spacing among all clusters. The vertical coordinate of a vertex is equal to the one assigned to its layer, which is proportional to the index of that layer in the total order of layers.

Going back to the state-of-the-art, concerning restrictions $R1$, $R2$ and $R3$, described in [16], that a planar clustered hierarchical drawing must obey, drawings produced by our algorithm satisfy $R1$ and $R2$, while we consider $R3$ too restrictive for our application. $R1$ is satisfied in the preprocessing step by merging, for each cluster, all sources into one vertex. The PQ-tree is initialized with a spanning tree that contains all these new vertices, which has the effect to keep the vertices of each cluster consecutive on any layer. $R2$, as shown in [16], is automatically satisfied for the initial drawing Γ_0, and is satisfied for any drawing of $G_{t,\mathcal{E}}$ by exploiting the partial order $\prec$.

To obtain a radial drawing, the geometric coordinates of vertices so computed are transformed as follows. Each vertex is placed on the perimeter of a circle centered in an arbitrary fixed point and having radius equal to the vertical coordinate of the vertex. Then the horizontal coordinate of the vertex is mapped to a circular coordinate on the perimeter of that circle. The perimeters of clusters are mapped with a similar radial transformation. An edge (u, v) is drawn either as a straight segment or a curved arc, depending on the angle it must sweep to connect vertices u and v. Note that in our setting each edge connects only vertices in two consecutive layers, hence a curved edge can be drawn only in the space between these layers.

4 Implementation and Technical Challenges

The implementation of TPLAY is split into three main blocks: 1. a visualization front-end; 2. a layout engine; and 3. a data back-end.

The *visualization front-end* is a Web application. It allows the user to specify input parameters and to visualize and animate interactive graphs. The main interface is presented in Fig. 1 and additional images are provided at [8]. It is composed of four main elements: the controller, the graph panel, the info panel, and the timeline panel. We detail their functionalities below.

The *controller* is a sliding panel located in the upper right corner. It allows the user to input queries composed of a target τ, a time interval $\mathcal{T}$, and a set of probes $\mathcal{S}$. Once the visualization is ready, the controller can be used to animate the graph with the traceroute

paths available during $\mathcal{T}$. The play/pause/stop, repeat-last, step-back, and step-forward buttons allow for a fine-grained management of the graph animation.

The *graph panel* displays the interactive graph, initially centered and fitted to the window. The user can pan and zoom it with the mouse. The animation of the graph consists of a sequence of morphing steps. Each step transform the graph by applying the effects of an event involving one or more traceroute paths. Given a probe $\sigma \in \mathcal{S}$, an event can consist of: (a) the availability of a new traceroute path from σ to τ; (b) a change in the sequence of vertices in the traceroute path from σ to τ; (c) a disconnection resulting in an empty traceroute path. Events of type (a) are rendered with a gradual introduction of new paths in the graph. Events of type (b) are rendered with a geometric morph of curves representing the involved paths. Events of type (c) are rendered with a blinking effect after which paths disappear. We introduce a delay between each pair of consecutive animation steps. The delay is proportional to the logarithm of the elapsed time between the corresponding routing events. This gives an approximate perception of elapsed time, while limiting the overhead on the total animation time. The elements of the graph are interactive and show additional information on request. Hovering a vertex with the mouse for a few seconds highlights all the paths passing through it. Hovering a path for a few seconds highlights the path and all its vertices.

The *info panel* is in the upper part of the window. It shows all the available information about any selected network component represented in the graph. It also displays a textual description for the latest event that caused an update of the visualized graph.

The *timeline panel* is in the lower part of the window and contains two timelines that allow to accurately navigate the traceroute information in $\mathcal{T}$. The first, called *control timeline*, provides a fast overview of the trend in the number of events over time. The second, called *selection timeline*, shows individual events ordered in time and is designed for fine-grained analysis. Each block in the selection timeline contains a sequence of events happening at the same time, represented with colored rectangles. Different colors are used for different types of events. The elapsed time between any two consecutive blocks is reported in the area between them. Both timelines feature a red cursor that points at the current time instant and is continuously updated during the animation. The user can drag the cursors, changing the current instant and updating the graph accordingly. The selection timeline can only show a limited number of events due to its constrained area. In case there are more events, the animation causes involved events to be smoothly translated into the visible part of selection timeline. The user can scroll horizontally to reveal hidden events. Further, the user can limit the animation to a particularly interesting period within $\mathcal{T}$ by dragging the two green sliders at the top of the control timeline. The sliders on the selection timeline are updated accordingly.

The implementation of the front-end required to focus on some algorithmic details. The arrangement of paths in a metro-line fashion is implemented as follows. First of all, an arbitrary total ordering is computed on the set of visualized paths. For each edge without bends in the graph, the paths that traverse it are drawn as parallel segments connecting the two endpoints of the edge. The ordering of such segments reflects the total ordering of paths to promote consistency between edges. In case the edge contains bends, the drawing is computed in two steps. First, we split the bended edge in a sequence of intermediate edges $e^1, \ldots, e^n$ and compute the path segments for each of

them. Second, for each pair of consecutive intermediate edges (e^i, e^{i+1}) and for each path that traverses it, we call (u, v) and (w, z) the two segments computed respectively for e^i and e^{i+1}. If there is an intersection point p between (u, v) and (w, z), we rewrite the two segments as (u, p) and (p, z). Otherwise, we add a connection (v, w) between (u, v) and (w, z). Path colors are computed with the algorithm described in [19] to ensure that they are distinguishable from each other.

The front-end is written in JavaScript and HTML. It is based on the BGPlay.js framework [1] that we developed in collaboration with the RIPE NCC. The objective of the framework is to simplify the implementation of web-based tools for the representation of evolving data described in terms of graph components. The framework consists of a solid implementation of graph domain objects and a set of modules. Modules provide functionalities and representation of data and can be used to compose ad-hoc tools. We use Scalable Vector Graphics for the representation and animation of the graph.

The visualization always starts with an overview of the traceroutes. Hence, the *layout engine* is invoked to produce a drawing of $G_{\mathcal{T},\emptyset}$. When the user expands/contracts a cluster (a cluster is added/removed from $\mathcal{E}$) the layout engine is invoked again on $G_{\mathcal{T},\mathcal{E}}$. In the implementation of the radial drawing we artificially increase the radius of each layer by an additional offset, such that vertices on dense layers are not overlapped. For the sake of simplicity, curved segments are uniformly sampled and drawn as polylines.

The layout engine is implemented in Java. We initially designed it to be implemented as part of the visualization front-end, but later moved to a back-end implementation in order to make use of already existing libraries. In particular we adopted a PQ-tree implementation [17] and Apache Commons Graph [9] for general graph models and algorithms. We optimized the output of the layout engine after the initial layout, so that only graph elements with new drawing coordinates are included.

The *data back-end* is mainly responsible of retrieving and preprocessing traceroute data. The result is then used by the front-end to animate traceroute events and by the layout engine to compute the drawings.

5 Conclusions and Open Problems

We presented a metaphor for the visualization of traceroute measurements towards specific targets on the Internet. It consists of a radial drawing of a clustered graph where vertices are routers or computers and clusters are administrative authorities that control them. Our metaphor allows the user to interact with the visualization, both exploring the content of individual clusters and animating the graph to see how traceroute paths change over a time interval of interest.

In the future we will take into account the *DNS resolution* of selected targets in the visualization. That means that some targets may be represented by more than one vertex, giving rise to an anycast behavior of the target, depending on the policies implemented at the DNS level. We will also explore the possibility to process streams of incoming data, adding or removing elements in the visualization incrementally.

References

1. BGPlayJS, `http://www.dia.uniroma3.it/ compunet/www/view/tool.php?id=bgplayjs`
2. CAIDA Ark, `http://www.caida.org/projects/ark/`

3. Measurement Lab, http://www.measurementlab.net/
4. MisuraInternet, https://www.misurainternet.it/
5. Monitor Scout Traceroute, http://tools.monitorscout.com/traceroute/
6. RFC 1918, address allocation for private internets, http://www.ietf.org/rfc/rfc1918.txt
7. SamKnows, http://www.samknows.com/broadband/
8. TPlay, http://www.dia.uniroma3.it/~compunet/projects/tplay
9. Apache Software Foundation. Apache Commons Graph, http://commons.apache.org
10. Augustsson, B.: Xtraceroute, http://www.dtek.chalmers.se/~d3august/xt/index.html
11. Bachmaier, C.: A radial adaptation of the sugiyama framework for visualizing hierarchical information. IEEE Trans. on Visualization and Computer Graphics 13(3), 583–594 (2007)
12. Di Battista, G., Didimo, W., Marcandalli, A.: Planarization of clustered graphs. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 60–74. Springer, Heidelberg (2002)
13. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. JCSS 13(3), 335–379 (1976)
14. Colitti, L., Di Battista, G., Mariani, F., Patrignani, M., Pizzonia, M.: BGPlay: A System for Visualizing the Interdomain Routing Evolution. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 295–306. Springer, Heidelberg (2004)
15. Di Battista, G., Nardelli, E.: Hierarchies and planarity theory. IEEE Transactions on Systems, Man and Cybernetics 18(6), 1035–1046 (1988)
16. Forster, M., Bachmaier, C.: Clustered level planarity. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 218–228. Springer, Heidelberg (2004)
17. Harris, J.: A graphical Java implementation of PQ-Trees, http://www.jharris.ca
18. Hokstad, V.: Traceviz: Visualizing traceroute output with graphviz, http://www.hokstad.com
19. Kistner, G.: Generating visually distinct colors, http://phrogz.net/css/distinct-colors.html
20. Periakaruppan, R., Nemeth, E.: Gtrace - a graphical traceroute tool. In: Proc. 13th USENIX Conference on System Administration, pp. 69–78. USENIX Association (1999)
21. Raitner, M.: Visual navigation of compound graphs. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 403–413. Springer, Heidelberg (2005)
22. RIPE NCC. RIPE Atlas, http://atlas.ripe.net/
23. RIPE NCC. RIPEstat, https://stat.ripe.net/
24. Roberts, M.J.: Underground Maps Unravelled - Explorations in Information Design (2012)
25. Sander, G.: Layout of compound directed graphs. Technical report, FB Informatik, Universitat Des Saarlandes (1996)
26. Sander, G.: Graph layout for applications in compiler construction. Theoretical Computer Science 217(2), 175–214 (1999)
27. Sugiyama, K., Misue, K.: Visualization of structural information: automatic drawing of compound digraphs. IEEE Trans. on Systems, Man and Cybernetics 21(4), 876–892 (1991)
28. Sundaresan, S., de Donato, W., Feamster, N., Teixeira, R., Crawford, S., Pescapè, A.: Broadband internet performance: A view from the gateway. In: Proc. SIGCOMM (2011)
29. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall (1998)
30. Visualware. VisualRoute, http://www.visualroute.com/
31. Yee, K.-P., Fisher, D., Dhamija, R., Hearst, M.: Animated exploration of dynamic graphs with radial layout. In: Proc. INFOVIS 2001. IEEE Computer Society (2001)